

Balanced offline allocation of weighted balls into bins

Ömer Demirel, Ivo F. Sbalzarini

*MOSAIC Group, Max Planck Institute of Molecular Cell
Biology and Genetics, Center of Systems Biology
Pfotenhauerstr. 108, D-01307 Dresden, Germany.*

Abstract

We propose a sorting-based greedy algorithm called **SortedGreedy**[**m**] for approximately solving the offline version of the d -choice weighted balls-into-bins problem where the number of choices for each ball is equal to the number of bins. We assume the ball weights to be non-negative. We compare the performance of the sorting-based algorithm with a naïve algorithm called **Greedy**[**m**]. We show that by sorting the input data according to the weights we are able to achieve an order of magnitude smaller *gap* (the weight difference between the heaviest and the lightest bin) for small problems (≤ 4000 balls), and at least two orders of magnitude smaller *gap* for larger problems. In practice, **SortedGreedy**[**m**] runs almost as fast as **Greedy**[**m**]. This makes sorting-based algorithms favorable for solving offline weighted balls-into-bins problems.

Keywords: Balls-into-bins, load balancing, offline algorithm, sorting.

1 Introduction

The classical *balls-into-bins* problem [8, 9] considers the sequential placement of n balls into m bins such that the bins are maximally balanced. Historically, the problem is categorized by the types of balls (e.g., uniform [1, 2] vs. weighted [4, 15, 12, 6]), by the number of bins a ball can choose from (e.g., single-choice vs. multi-choice [10]), and by the number of balls (e.g., $n = m$ [13] vs. $n > m$ or $n \gg m$ [3]). In applications such as load balancing, hashing, and occupancy problems in distributed computing [4, 3, 5] the d -choice variant and its subproblem, the two-choice variant have been the main focus. This is because the mathematical analysis of load-balancing algorithms frequently involves solving a *balls-into-bins* problem where the balls and bins represent tasks and compute units, respectively.

Of special practical importance is the weighted case where the balls have individually different weights. Talwar and Wieder [15] have shown that as long as the weight distribution has finite second moment, the weight difference between the heaviest and the average bin (i.e., the *gap*) is independent of n . Peres *et al.* [12] introduced the $(1 + \beta)$ -choice process analysis, and for $\beta = 1$ the *gap* has a bound $\Theta(\log \log n)$ even for the case of weighted balls. Dutta *et al.* [6] introduced the *IDEA* algorithm, which provides a constant *gap* with high probability (w.h.p.) even in the heavily loaded

case $n \gg m$ in case of an expected constant number of retries or rounds per ball.

The *offline* version of the weighted balls-into-bins problem has received less attention than the *online* version. We believe, however, that the offline version is as important in practice as the online version, since in compute systems with a priorly known number of tasks the optimal assignment of these tasks to processors and the expected load imbalance can be analyzed by solving an offline balls-into-bins problem. In the offline setting, we define the *gap* as the weight difference between the heaviest and the lightest bin. We do not restrict the distribution from which the balls sample their weights. For simplicity, we assume that a ball can be placed into any bin, thus $d = m$. We propose to initially sort the balls according to their weights and then use a greedy algorithm to place the next heaviest ball into the lightest bin. We show that even for moderate problem sizes ($n < 4000$ balls) this sorting-based greedy algorithm results in a 10 to 60-fold smaller *gap* than the naïve **Greedy** [2] algorithm. Furthermore, we show using simulations that the gap resulting from the present sorting-based algorithm decreases exponentially with increasing n . Moreover, the time overhead due to sorting is negligible, which makes the sorting-based algorithm also practically useful.

This paper is structured as follows: Section 2 introduces the notation. In section 3 we introduce two sorting-based algorithms: **SortedGreedy** [2] and a distribution-based sorting algorithm. We investigate their theoretical time complexities and compare the results with **Greedy** [2] in Section 4. Finally, Section 5 concludes the paper with a discussion and notes on future work.

2 Notation

We are given a set of n weighted balls W_i and m bins. The total weight of a bin U_i is given by the sum of the weights of the balls it contains after all balls are assigned. Since we know the ball weights *a priori*, we can easily compute the ideal (but impossible since the balls are indivisible) total weight of each bin: W_t/m , where $W_t = \sum_{i=1}^n W_i$. The task is to place all n balls sequentially into the m bins such that the *gap* $G = \max_i U_i - \min_i U_i$ is minimized. We make no assumption about the distribution from which the balls sample their weights, unless otherwise mentioned. The standard deviation of *gap* is denoted by σ .

3 Algorithms

The goal is to minimize the gap in the m -bin case, where $m \geq 2$. Below we use two greedy algorithms to approximately solve this problem. We compare them with each other both theoretically and in numerical experiments.

3.1 Greedy [2] algorithm

The online version of the **Greedy** [2] algorithm has previously been proposed [1, 2] and extended to the weighted balls case [15]. Talwar *et al.* have shown that the weight difference between the average and heaviest bin is independent of n . The only modification to the problem in the offline version is that we are given the n balls *a priori*. The algorithm chooses a ball W_i and places it into the lightest bin. Ties are broken arbitrarily. The algorithm is repeated until all balls have been assigned.

The time complexity of this algorithm is $\Theta(n)$, since we go through all balls exactly once. The pseudo-code of this algorithm is given in Algorithm 3.1.

Algorithm 3.1: GREEDY[M]($U_{1...m}, W$)

comment: Given are a set W of n balls and the bin arrays $U_{1...m}$

comment: Assign the first value to the first bin

$U_1[1] \leftarrow W[1]$

comment: Initialize the pointers for all bins

$p[2...m] \leftarrow 1$

comment: First bin has already one ball in it.

$p[1] \leftarrow 2$

comment: Give remaining $n - 1$ balls sequentially to lightest bin

for $i \leftarrow 2$ **to** n

do $\left\{ \begin{array}{l} \textbf{comment:} \text{ Find the ID of the lightest bin which is the one with least current sum} \\ idx \leftarrow \text{findLightestBin}(U_{1...m}) \\ U_{idx}[p_{idx}] \leftarrow W[i] \\ p_{idx} \leftarrow p_{idx} + 1 \end{array} \right.$

return ($U_{1...m}$)

3.2 Sorting-based algorithms

Sorting-based algorithms consist of two phases: sorting and greedy placement. The latter then amounts to applying **Greedy**[m] to balls sorted in the order of descending weights, such that $W_1 \geq W_2 \geq \dots, W_{n-1} \geq W_n$. Starting from W_1 all balls are thrown sequentially into the bin with least current total weight.

In addition to finding the best-balanced allocation, it is also important to devise practically usable sorting-based algorithms. This can be accomplished by exploiting any given information about the problem. For instance, depending on the available knowledge about the weight distribution, we can propose two different sorting strategies. Regardless of which sorting strategy is chosen and which sorting algorithm is used, however, the resulting *gap* is the same for all sorting-based algorithms. The pseudo-code of a sorting-based algorithm called **SortedGreedy**[2] is shown in Algorithm 3.2.

Algorithm 3.2: SORTEDGREEDY[M]($U_{1...m}, W$)

comment: Given are a set W of n balls, and the bin arrays $U_{1...m}$

comment: Sort the array in descending order (e.g. using quicksort)

$sortedW \leftarrow \text{quicksort}(W)$

return (GREEDY[M]($U_{1...m}, sortedW$))

3.3 Uniform weight distribution

If the weights are sampled from a uniform distribution over the interval $[0, A]$, $A \in \mathbb{R}^+$, we can use a distribution-based sorting algorithm, such as bucketsort, Proxmap-sort [14], or flashsort [11]. Since these algorithms are not comparison-based, the $\Omega(n \log n)$ lower bound for comparison-based sorting does not apply to them. For example, Proxmap-sort [14] has an average time complexity of $O(nk) = O(n)$, where $k < n$ is the content number of “buckets” used for sorting. Thus, the algorithm outperforms the lower bound for comparison-based sorting for large n . The worst-case complexity of distribution-based sorting algorithms, however, is $O(n^2)$ as n approaches k . However, the probability of the worst case scenario (i.e., having $k = n$ buckets) is small since k is user-defined. For flashsort $k = 0.42n$ has been found a good value in empirical tests [11].

3.4 Other distributions

For non-uniform weight distributions, we resort to efficient comparison-based sorting algorithms, such as mergesort or quicksort [7]), which have an average time complexity in $O(n \log n)$. Depending on the specific sorting algorithm, the worst-case complexity can also be in $O(n \log n)$. Highly optimized implementations of these algorithms are commonly available, rendering them useful in practice.

4 Simulation results

We implement both **Greedy**[2] and **SortedGreedy**[2] in MATLAB (R2012a, The Mathworks, Inc., Natick, MA, USA). **SortedGreedy**[2] uses MATLAB’s intrinsic quicksort function to sort the balls according to their weights. The balls are assigned random weights sampled from a uniform distribution over the interval $[0, 10]$. Each simulation is repeated 1000 times with different random weights, and we report the mean and standard deviation of the gap for different numbers of balls and bins.

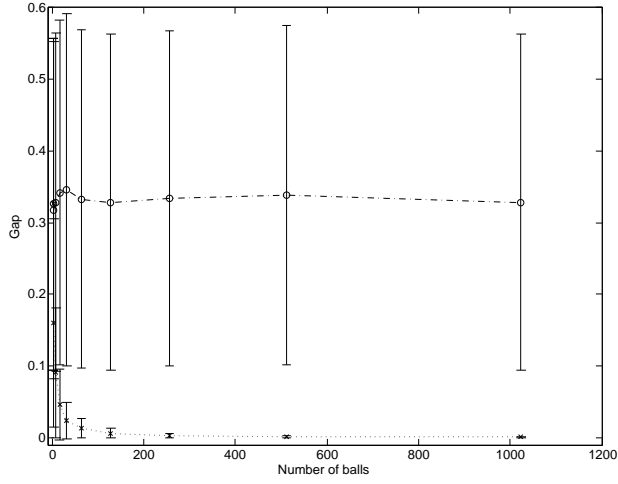
4.1 Increasing n

Figure 1 shows the results for $m = \{2, 8\}$ bins and varying numbers of balls. The σ bars for **Greedy**[2] are independent of n with $\sigma = 0.23$ for $m = 2$ and $\sigma = 0.15$ for $m = 8$. For **SortedGreedy**[2] the average σ is 0.01 for $m = 2$ and 0.03 for $m = 8$.

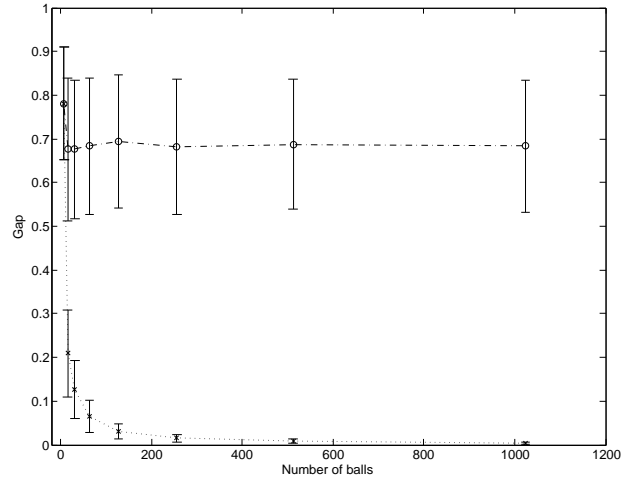
As seen in Fig. 1, **SortedGreedy**[2] outperforms **Greedy**[2] in all tested cases, including those with odd numbers of balls. The *gap* resulting from **SortedGreedy**[2] decreases exponentially as the number of balls increases, and it is at least 10 times smaller than the gaps obtained by **Greedy**[2] when $n \gg m$. For each m -bin problem, the standard deviation across the random repetitions of the **Greedy**[2] algorithm remains constant. Also, the *gap* resulting from **Greedy**[2] remains almost constant with n .

4.2 Increasing m

In Fig. 2 we show the dependence of the *gap* on the number of bins m for $n = \{1024, 3027\}$. The gap obtained by **Greedy**[m] first increases rapidly and then seems to saturate. That from **SortedGreedy**[m] initially increases much slower. This is in line with previous findings [15]. Indeed,

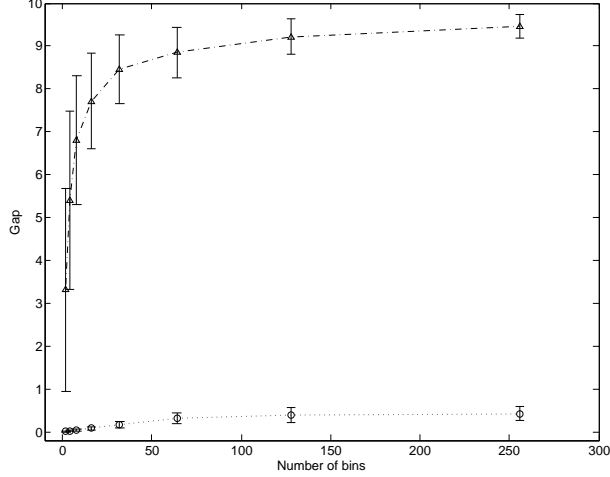


(a) $m = 2$

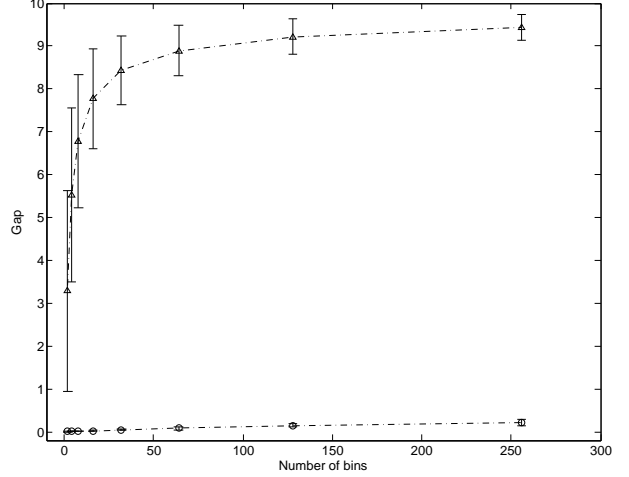


(b) $m = 8$

Figure 1: The *gap* is shown for each n . On average, the gaps achieved by **SortedGreedy**[2] (‘ \circ ’) are an order of magnitude smaller than those obtained by **Greedy**[2] (‘ \triangle ’). (a) The case for $m = 2$ bins. For $n \geq 32$, the average *gap* ratio between the two algorithms increases to 60. (b) The case for $m = 8$ bins. Here, the *gap* ratio is about 73 for $n \geq 512$.



(a) $n = 1024$



(b) $n = 3027$

Figure 2: The *gap* achieved for different numbers of bins and a constant number of balls: (a) 1024 balls, (b) 3027 balls. The results are shown for the **SortedGreedy**[**m**] algorithm (‘ \circ ’) and the **Greedy**[**m**] algorithm (‘ \triangle ’).

Talwar *et al.* [15] show that the *gap* depends on both the distribution from which the weights are sampled, and on m .

4.3 Timings

We perform runtime measurements for the two-bin problem with $n = 2^{13}$. The experiment is repeated 100 times and averages are recorded. All test runs are conducted on a Macbook Pro (MacOS X 10.7.5) with a quad-core 2.3 GHz Intel Core i7 processor and 8 GB 1600 Mhz DDR3 memory. Both algorithms require approximately the same time to solve the two-bin problem. For placing 2^{13} balls 0.1950 s are needed by **SortedGreedy**[2] and 0.1948 s by **Greedy**[**m**]. Thus, sorting adds an overhead of about 2 ms, which is 0.02% of the total runtime. Increasing m has no substantial effect on the final runtime as long as $n \gg m$.

5 Discussion

We outlined two algorithms, `Greedy[m]` and `SortedGreedy[m]`, to solve the offline version of the weighted balls-into-bins problem. We compared their asymptotic time complexities and simulation performances. `SortedGreedy[m]` finds at least an order of magnitude better *gaps* compared to `Greedy[m]` for $m \leq 32$ and $n \gg m$. The difference grows with increasing problem size. For $n \geq 4096$ the *gap* ratio is at least two orders of magnitude in favor of `SortedGreedy[m]`.

The *gap* from `SortedGreedy[m]` decreases exponentially with increasing numbers of balls. Moreover, `SortedGreedy[m]` is only weakly affected by increasing numbers of bins. The time complexity of `Greedy[m]` is in $O(n)$. When the balls sample their weights from an uniform probability distribution, `SortedGreedy[m]` has the same asymptotic time complexity. For other weight distributions, the runtime of `SortedGreedy[m]` is in $O(n \log n)$. As shown by our numerical experiments, however, this only incurs a minor toll in practice and both algorithms execute in almost identical times. Therefore, we conclude that the `SortedGreedy[m]` algorithm is favorable for approximately solving instances of the offline weighted balls-into-bins problem in practice.

Future work will consider the design of a distributed dynamic load balancing protocol based on `SortedGreedy[m]`. Such a protocol could be used in high-performance computing system for more efficiently solving task-to-processor assignment problems arising in real-world applications.

Acknowledgements

We thank all members of the MOSAIC Group (MPI-CBG, Dresden) and Dr. Erdem Yörük (Johns Hopkins University) for many fruitful discussions.

References

- [1] Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal, *Balanced allocations*, Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, ACM, 1994, pp. 593–602.
- [2] ———, *Balanced allocations*, SIAM journal on computing **29** (1999), no. 1, 180–200.
- [3] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking, *Balanced allocations: The heavily loaded case*, SIAM Journal on Computing **35** (2006), no. 6, 1350–1385.
- [4] Petra Berenbrink, Tom Friedetzky, Zengjian Hu, and Russell Martin, *On weighted balls-into-bins games*, Theoretical Computer Science **409** (2008), no. 3, 511–520.
- [5] Artur Czumaj, Chris Riley, and Christian Scheideler, *Perfectly balanced allocation*, Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques, Springer, 2003, pp. 240–251.
- [6] Sourav Dutta, Souvik Bhattacharjee, and Ankur Narang, *Perfectly balanced allocation with estimated average using approximately constant retries*, CoRR, vol. abs/1111.0801 (2011).
- [7] Charles AR Hoare, *Quicksort*, The Computer Journal **5** (1962), no. 1, 10–16.
- [8] Norman Lloyd Johnson and Samuel Kotz, *Urn models and their application: an approach to modern discrete probability theory*, Wiley New York, 1977.
- [9] Valentin Fedorovich Kolchin, Boris Aleksandrovich Sevastianov, and Vladimir Pavlovich Chistiakov, *Random allocations*, Vh Winston New York, 1978.
- [10] Michael Mitzenmacher, *The power of two choices in randomized load balancing*, Parallel and Distributed Systems, IEEE Transactions on **12** (2001), no. 10, 1094–1104.
- [11] Karl-Dietrich Neubert, *“flashsort”*, Dr. Dobbs Journal (1998).
- [12] Yuval Peres, Kunal Talwar, and Udi Wieder, *The $(1 + \beta)$ -choice process and weighted balls-into-bins*, Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2010, pp. 1613–1619.
- [13] Martin Raab and Angelika Steger, *“balls into bins”—a simple and tight analysis*, Randomization and Approximation Techniques in Computer Science, Springer, 1998, pp. 159–170.
- [14] Thomas A Standish, *Data structures in java*, Addison-Wesley Longman Publishing Co., Inc., 1997.
- [15] Kunal Talwar and Udi Wieder, *Balanced allocations: the weighted case*, Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, ACM, 2007, pp. 256–265.